Adhoc 云中基于移动预测的多准则任务卸载算法 *

田广东，葛东玉

(重庆邮电大学 通信与信息工程学院，重庆 400065)

摘 要：为了提升 Ad hoc 云中任务卸载的效率，针对节点随机移动性及资源异构性对任务卸载的影响，提出一种基于移动预测的多准则任务卸载算法，根据时间序列分析预测节点逃离时间，并将其作为节点移动性衡量指标。运用层次分析法得到 CPU 速度、核心数、负载及移动性的权重，最后根据任务大小及计算得到的组合权重进行任务卸载，仿真结果表明，相比于随机任务分配算法和 Min-Min 调度算法，该算法能够有效降低任务执行时间和能量消耗。

关键词：Ad hoc 云；多准则；任务卸载；时间序列分析；层次分析法

中图分类号：TN92 doi: 10.19734/j.issn.1001-3695.2018.07.0468

Multi criteria task offloading algorithm in Ad hoc cloud based on mobility prediction

Tian Guangdong, Ge Dongyu

(*School of Communication & Information Engineering*, *Chongqing University of Posts & Telecommunication*, *Chongqing* 400065, *China*)

Abstract: In order to improve the efficiency of task offloading in Ad hoc cloud, aiming at the effect of node random movement and resource heterogeneity on task offloading, this paper proposes the multi criteria task offloading algorithm based on mobility prediction, which is used to predict node escaping time based on time series analysis and as a measure of mobility. By using the analytic hierarchy process to get the weights of CPU speed, number of cores, workload and mobility, the task is offloaded according to task size and the combined weights which has been calculated, the simulation results show that the algorithm can reduce the task execution time and energy consumption effectively compared with the random task assignment algorithm and Min-Min scheduling algorithm.

Key words: Ad hoc cloud; multi criteria; task offloading; time series analysis; analytic hierarchy process

0 引言

随着智能设备的迅速普及，越来越多需要密集计算资源且高能耗的移动应用程序被开发应用，并受到广泛关注[1]。尽管近年来智能设备的性能得到了很大提升，但仍不能满足用户在大数据时代爆炸式的数据处理需求。因此，移动云计算[2]（mobile cloud computing，MCC）逐渐成为解决智能设备资源受限的主要技术。MCC 为智能设备提供了高效的远程计算服务，移动用户通过 MCC 将计算密集型的任务卸载到远程云或本地小型云设备[3]处理，完成后再将计算结果返回，以此降低任务在本地处理的开销，缓解移动设备的资源匮乏问题。远程云虽然计算能力强，但会带来延迟和开销较高的问题；本地小型云计算能力强且通信带宽大，但却无法保证用户随时随地接入；而 Ad hoc 云[4]作为无基础设施下 MCC 的一种扩展，可以通过与附近的移动设备共享其闲置资源进行任务卸载[5]，具有部署快速且扩展灵活等优点。

为实现高效的任务卸载，现有研究为 MCC 设计了各种协议与机制，而对于 Ad hoc 云的研究仍处于起步阶段，为解决计算密集型应用卸载问题，文献[6]提出了一种在线批调度启发式方法，动态地选择移动设备进行任务卸载。文献[7]提出了一种用于异构移动云的通用卸载框架，它使用移动设备、附近的小型云和远程云来提高 MCC 服务的性能和可用性。然而，所提出的框架并没有充分考虑 Ad hoc 云的特性。

为了满足移动设备的延迟限制，文献[8]提出了一种节能任务调度方案，并针对不同的任务，设计了一种自适应概率调度器，不仅能满足延迟约束，而且能在执行计算密集型实时应用程序时最小化能耗，但其计算复杂度较高。为降低能耗和计算成本，文献[9]提出了一种基于微云辅助的任务分配机制，制定了一个两阶段 Stackelberg 博弈来确定 slave 节点提供的执行单元数量，而 master 节点将以此确定补偿策略，但是该机制在任务分配时并未考虑可用移动设备的资源异构性。

上述方法虽然能够在一定程度上降低任务执行的时间或能

收稿日期：2018-07-13；**修回日期：**2018-08-22 **基金项目：**安徽省科技攻关计划项目（1501zc03032）

作者简介：田广东（1968-），男，教授，硕导，主要研究方向为物联网与智慧城市（tiangd2005@163.com）；葛东玉（1993-），男，硕士研究生，主要研究方向为无线自组织网络.

耗，但是均未充分考虑 Ad hoc 云中节点的随机移动性和资源异构特性对任务卸载的影响。虽然影响任务卸载的因素众多，而作为 Ad hoc 网络最基本的特性，节点的随机移动性和异构性无疑是影响 Ad hoc 云中任务卸载的最重要因素，如何选择合适的节点及合理地分配资源成为亟待解决的问题。针对于上述问题，本文提出了一种基于移动预测的多准则任务卸载算法，充分考虑 Ad hoc 云中节点的随机移动性和资源异构性，提高任务卸载效率。

1 系统模型

为了不失一般性，假设在 Ad hoc 云中有 $M+1$ 个节点，其中的一个节点（master）有一组待处理的计算密集型任务 W，如果只依靠自身资源来完成这些任务很可能会耗尽自身资源或带来较高的延迟，十分影响用户体验。而其他 M 个节点（slave）则具备较为丰富的闲置资源，在一定范围内 master 节点可以通过优质的无线网络（Wi-Fi）将任务卸载到 slave 节点，任务完成后再将结果返回，从而实现自身资源的扩展。

本文主要考虑节点的移动性和异构性，暂不考虑节点间通信成本。为了实现多任务同时执行，假设 master 节点根据处理器时隙进行任务分配，则第 i 个任务的执行时间可表示为

$$T_i=\sum_{s=1}^{i}t_s=\sum_{s=1}^{i}(\frac{W_i}{S\times C}+\frac{D_i}{S\times C}) \tag{1}$$

即第 i 个任务的执行时间为其执行期间所有时隙的和，其中第一个时隙的大小取决于该时隙内执行任务的数量和最小任务的大小，而剩余时隙的大小取决于该时隙内当前执行任务和下一个较小任务大小的差异，即

$$t_i=\begin{cases}(\min(W)\times N)/(S\times C) & i=1\\(W_i-W_{i-1})\times(N-(i+1))/(S\times C) & i>1\end{cases} \tag{2}$$

其中：S 和 C 分别为节点 CPU 速度和核心数，D_i 为第 i 个任务执行时节点原有负载。

2 多准则任务卸载算法

节点的移动性和异构性作为 Ad hoc 网络最基本的特性，对 Ad hoc 云中高效的任务卸载有着重要影响，高效的任务卸载能显著降低任务执行时间和能量消耗，提升用户体验。本文综合考虑 Ad hoc 云中节点的移动性与资源异构性，提出一种高效的任务卸载算法，能够有效降低任务执行时间与能量消耗。

2.1 移动预测

在 Ad hoc 网络中，节点移动性的量化通常表示为相对移动速度，根据连续两次收到的信号强度差异确定节点移动速度，即

$$v_i=\lg\frac{P_{r2}}{P_{r1}} \tag{3}$$

其中：P_{r1} 和 P_{r2} 分别为连续两次接收的信号功率。显然，若 $v_i>0$，则认为节点在靠近，移动性相对较好；反之则认为其移动性较差。这种方法虽然计算简单，但实际网络中节点的能量会随着网络的运行而降低，并且信号接收功率很容易受噪声影响，存在较大误差。

而根据时间序列分析[10]可以较为准确地预测 Ad hoc 网络中节点的位置信息。任何实际的 Ad hoc 网络都是需要完成某些具体任务的，因此，节点的运动不会是完全杂乱无章的。如果将节点的移动坐标信息采样为离散时间序列，则可以利用时间序列进行建模和预测，使节点能够预测其他节点在未来一段时间内的位置信息，在一定程度可以减少网络的开销，提升资源利用率。

预测问题是随机过程领域的基本问题，针对时间序列的预测，指的是在 t 时刻对未来第 l 步的取值 x_{t+l} 进行预测，获得预测值 $x_t(l)$。研究表明，一个平稳的时间序列总可以表示成一系列白噪声 $\{A_t\}$ 的线性组合，即

$$x_t=\sum_{j=0}^{\infty}G_ja_{t-j} \tag{4}$$

令 $t=t+l$ 并展开得：

$$\begin{aligned}x_{t+l}=&(G_0a_{t+l}+G_1a_{t+l-1}+\cdots+G_{l-1}a_{t+1})\\&+(G_la_t+G_{l+1}a_{t-1}+\cdots G_{l+t-1}a_1)\end{aligned} \tag{5}$$

其中，对于右边第一部分，由于 $\{A_t\}$ 是白噪声，因而在 t 时刻无法预测未来 l 时间段内的序列值，即这一部分是无法计算的，可视为第 l 步的预测误差 $e_t(l)$。对于第二部分，由于在 t 时刻，所有的序列值 $a_t,a_{t-1},\ldots,a_1$ 都已确定，因此是可以计算的，可视为第 l 步的预测值 $x_t(l)$，则式(5)可记为

$$x_{t+l}=e_t(l)+x_t(l) \tag{6}$$

由于 $x_t(l)$ 包含了所有可计算的项，因此可以认为预测误差 $e_t(l)$ 的方差达到最小。设白噪声 $\{A_t\}$ 的方差为 σ_a^2，则 $e_t(l)$ 的方差为

$$Var[e_t(l)]=\sum_{j=0}^{l-1}G_j^2Var[a_{t+l-j}]=\sigma_a^2\sum_{j=0}^{l-1}G_j^2 \tag{7}$$

在几何上可以证明 $x_t(l)$ 为向量 x_{t+l} 在无限维上的正交投影，此时，$e_t(l)$ 为最小，即 x_{t+l} 完全回归于 $x_t(l)$。这就是使预测误差的方差为最小意义下的最优预测，即最小均方误差预测。

针对随机时间序列建模时，为确保可靠性，通常要求时间序列具有平稳性、正态性、零均值性等统计特性。因此，在获得样本序列后，首先需要检验该序列是否满足这些特性。如果不满足，则需要通过预处理得到满足要求的时间序列。

序列的平稳性检验可采用参数型检验法或非参数型检验法。对于非平稳时间序列，通常可采用多次差分的方式进行平稳化，并且只需在后续处理过程结束后进行恢复即可；正态性检验，也称为纯随机性检验。纯随机序列，即服从正态分布的白噪声，在统计学中被认为是不具有任何分析价值的。在平稳性检验后，

还需要检验正态性，确保该过程对应的时间序列具有进一步分析、预测的价值。对于一个具有实际价值的 Ad hoc 网络，其节点的移动总是为了完成某种特定任务。因此，节点的坐标序列不会是纯随机序列，可省略正态性检验的过程；零均值检验是检验时间序列所描述的随机过程均值是否为零。这里，时间序列描述的是节点的坐标，难以保证移动的节点坐标均值为零，因此需要对原始序列进行零化处理。

样本序列经过预处理后，就可以进行模型选择了。在时序分析预测领域，最常用的一组模型是 ARMA(Auto Regressive Moving Average Model)模型，这也是当前时序分析中效果最佳的一组预测模型。

模型选定之后还要进行参数估计和模型检验，ARMA 模型的参数估计方法大致上可以分为三类：第一类是基于时序理论本身所发展的参数估计法，第二类是基于优化理论迭代运算的优化参数估计法，第三类是基于控制理论中差分模型的参数估计法。这里主要使用第三类方法中的广义最小二乘估计。ARMA 的建模过程是一个动态修正的过程，在完成参数估计后，需要检验模型的有效性。理论上，ARMA 模型成立的根本条件是 $\{a_t\}$ 为白噪声，因此，实际使用的模型检验方法都是围绕这一点展开的。

根据以上步骤，结合具体的 Ad hoc 网络节点移动场景，给出预测流程：

a)初始化全局参数；

b)对节点的位置信息采样，得到样本序列；

c)判断序列是否平稳，若是则进行下一步，否则进行差分处理后再执行步骤 c)；

d)选择合适的时间序列预测模型；

e)根据选择的预测模型进行参数估计；

f)模型检验，判断预测误差是否为白噪声，若是则进行下一步，否则返回步骤 d)；

g)根据以上步骤进行最后的最小均方误差预测并输出结果。

如前所述，在实际的 Ad hoc 网络中，不会存在运动完全杂乱无章的节点。因此根据时间序列预测，基于节点的历史位置坐标，可以估算出其下一时刻的速度，再结合预测出的下一时刻位置坐标，可以估计各节点逃离 master 节点通信范围所需的时间 T_i^{pre}。显然，T_i^{pre} 越大，任务执行成功的概率就越高，移动性越好。假设节点在某时刻的移动轨迹如图 1 所示，以 master 节点坐标为圆心、通信距离 R 为半径构造其通信范围，图中 x_i 和 y_i 为节点位置坐标，v_x 和 v_y 分别为当前速度的水平和垂直分量。

则下一时刻节点的坐标为

$$x_t = \frac{-k(y_i - kx_i) \pm \sqrt{(1+k^2)R^2 - (kx_i - y_i)^2}}{1+k^2} \tag{8}$$

$$y_t = k(x_t - x_i) + y_i \tag{9}$$

其中 $k = v_y / v_x$，因此可以计算出 T_i^{pre} 为

$$T_i^{pre} = \frac{\sqrt{(x_i - x_t)^2 + (y_i - y_t)^2}}{\sqrt{v_x^2 + v_y^2}} \tag{10}$$

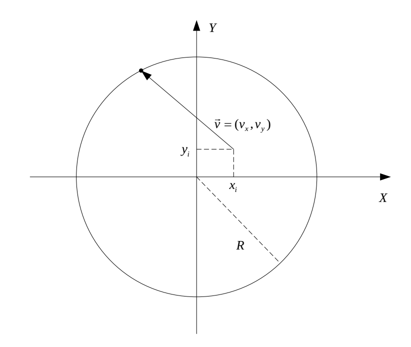


图 1　节点移动轨迹示意图

Fig.1　Node moving trajectory chart

2.2　权重确定

影响任务执行的异构因素主要有 CPU 速度、核心数及负载，根据以上因素和节点的移动性构造权值公式为

$$f_i = \omega_1 d_i + \omega_2 s_i + \omega_3 c_i + \omega_4 t_i \tag{11}$$

$$\omega_1 + \omega_2 + \omega_3 + \omega_4 = 1 \tag{12}$$

其中:ω_1，ω_2，ω_3，$\omega_4 \in (0,1)$ 为权值因子；s_i、c_i、d_i 和 t_i 分别表示节点 CPU 速度、核心数、剩余空闲负载和逃离时间。

本文采用层次分析法[11]确定权值因子，层次分析法是美国运筹学家 Saaty 教授提出的一种简便、灵活而又方便使用的多准则决策方法。该方法中各项比例标度及其意义如表 1 所示。

表 1　标度值及其意义

Table 1　Importance scale and their definitions

标度值	意义
1	两因素同等重要
3	一个因素稍微重要
5	一个因素明显重要
7	一个因素强烈重要
9	一个因素极端重要
2,4,6,8	为上述相邻判断的中值

多数公有云服务提供商都能为用户提供无限的资源，而在 Ad hoc 云中，可用资源高度依赖于节点数量及其硬件规格，节点的移动性则是另一个需要考虑用于任务卸载决策的因素，更接近的节点，往往可以被认为更适合任务卸载[12]，各因素对比情况如表 2 所示，其中节点的移动性可以用得到的预测逃离时间作为衡量指标，并且表中数据可根据实际情况作出相应调整。

表 2　各因素两两对比情况

Table 2　Various factors comparison levels

因素	CPU 速度	核心数	负载	移动性
CPU 速度	1	1	1/4	5
核心数	1	1	1/3	5
负载	4	3	1	7
移动性	1/5	1/5	1/7	1

根据表 2 可以得到判断矩阵为

$$A=\begin{pmatrix} 1 & 1 & 1/4 & 5 \\ 1 & 1 & 1/3 & 5 \\ 4 & 3 & 1 & 7 \\ 1/5 & 1/5 & 1/7 & 1 \end{pmatrix}$$

对该矩阵归一化处理后，通过计算得到各权值因子分别为 $\omega_1=0.548$，$\omega_2=0.194$，$\omega_3=0.206$，$\omega_4=0.062$。

2.3　算法描述

如前所述，影响任务卸载的因素主要有节点 CPU 速度、核心数、负载及移动性，综合考虑以上因素，结合时间序列分析与层次分析法提出一种多准则任务卸载算法，描述如下：

算法 1　基于移动预测的多准则任务卸载算法

INPUT : $N, S_N, W, C_N, P_N, CPI, R_T, \mathrm{R_N}, T_N^{pre}$

$X \leftarrow W$

Sort(Descend，X)

$\forall_{n=1\ldots N}\ E_{tL}^n = 0$ for i = 1 : $|X|$ do

$\forall n \in N\ map < X_i, E_L^n > \leftarrow \dfrac{|X_i| \times CPI_n / S_n}{R_{X_i^n}} \times 100$

end

$\forall n \in N, \mathrm{R_n} = P_n - E_{tL}^n$

while($|X|$!= NULL)

$(\hat{d_n}, \hat{s_n}, \hat{c_n}, \hat{t_n}) = \arg\max \mathrm{f}(d_n, s_n, c_n, t_n)$

$\forall n \in N, d_n \in \mathrm{R_N}, c_n \in C_N, t_n \in T_N$

$\mathrm{f}(d_n, s_n, c_n, t_n) \leftarrow \omega_1 d_n + \omega_2 s_n + \omega_3 c_n + \omega_4 t_n$

$NodeID \leftarrow getID(N, \hat{d_n}, \hat{s_n}, \hat{c_n}, \hat{t_n})$

$map < NodeID > \leftarrow x_1$ where $x_1 \in X$

$E_{tL}^n \leftarrow E_{tL}^n + E_L^n$

$X \leftarrow X / x_1$

end

OUTPUT：$map < N, X >$

其中: N 为节点数量，S_N 为节点 CPU 速度，W 为待卸载任务，C_N 为节点 CPU 核心数，P_N 为节点最大负载，CPI 为每条指令执行时 CPU 所花费的平均时钟周期数，R_T 为任务实际执行时间，$\mathrm{R_N}$ 为节点的剩余空闲负载，T_N^{pre} 为节点的预测逃离时间，E_{tL}^n 为节点预计执行任务时负载，上标 ∧ 表示 arg 函数的最大返回值。执行步骤如下：

a)master 节点通过周期性地广播“hello”报文发布任务卸载请求；

b)在通信范围内的节点根据自身状态选择是否接受任务，若接受则向 master 节点发送“hello”报文，报文携带的信息包括自身 CPU 速度、核心数、负载及位置信息；

c)master 节点根据任务大小将待卸载任务降序排列；

d)收集各节点提供的信息并根据位置信息计算其逃离时间；

e)根据式(11)和计算出的权值因子计算出各节点的组合权重；

f)将计算得到的组合权重值按照降序排列；

g)将任务列表中的任务按照对应关系分配给具有相应权重的节点；

h)判断任务完成情况，已完成任务移出列表，未完成任务重新分配；

i)不断重复以上步骤，直至任务列表为空或节点列表为空。

3　仿真与分析

本文通过 MATLAB 将所提 WCST 算法（即 Workload，Cores，Speed，Time）与随机任务分配算法（RM）及 Min-Min 调度算法[13]进行仿真对比。随机任务分配算法是将任务完全随机地分配给可用节点，而 Min-Min 调度算法则是网格计算中经典的任务调度算法，该算法首先找到全部任务的最小执行时间，然后在所有任务中选择具有最小执行时间的任务。算法通过将任务分配给能得到最小完成时间的资源而进行，重复相同的过程，直到全部任务都被调度完成。Min-Min 算法的局限性在于它优先选择较小的任务进行调度，使用了具有很高计算能力的资源，其结果是，当较小任务的数量超过较大任务的数量时，调度结果不会最佳，并且往往不能实现负载均衡。

仿真场景中选择 Ad hoc 网络中应用最广泛的随机点移动模型，节点数设置为 4，对应 CPU 分别为四核 1300MIPS、单核 1008MIPS、双核 1600MIPS 和八核 1200MIPS（Million Instructions Per Second，表示 CPU 每秒处理的百万级的机器语言指令数），待卸载任务设置如表 3 所示，其中每个任务分为五个子任务，通过六个任务的完成情况来验证所提算法的有效性，衡量指标为任务执行时间和能量消耗。

表 3　任务大小设置情况

Table 3　Task size setup situation

Task ID	子任务大小（指令数，单位：1.0e+11）				
1	0.0026	0.0452	0.2339	0.7392	1.1712
2	0.0018	0.0448	0.2366	0.7323	1.1190
3	0.0029	0.0438	0.2314	0.7321	1.2575
4	0.0016	0.0440	0.2347	0.7302	1.3371
5	0.0027	0.0438	0.2400	0.7311	1.4427
6	0.0028	0.0444	0.2335	0.7372	1.4018

任务执行时间是影响用户体验最重要的因素，图 2 是 RM 算法、Min-Min 算法以及 WCST 算法分别完成六个任务的执行时间对比图，从图中可以看出相比于 RM 算法，WCST 算法可以明显减少任务的执行时间，这是因为 RM 算法在任务分配时是完全随机的，而 WCST 算法充分考虑设备的异构性（CPU 速度、核心数及负载）和移动性，根据各因素的权重组合进行任务分配，保证了任务执行成功的可能，从而实现了高效的任务卸载。而相比于 Min-Min 算法，WCST 算法在任务 1 和任务 2 上消耗了稍长的执行时间，这是因为任务 1 和任务 2 较小，因此会被 Min-Min 算法优先调度并使用具有较强计算能力的资源，

而其余任务则依然是 WCST 算法更能有效降低任务执行时间，并且相比于 RM 算法和 Min-Min 算法，WCST 算法可以通过更合理的任务分配有效均衡各个任务的执行时间。

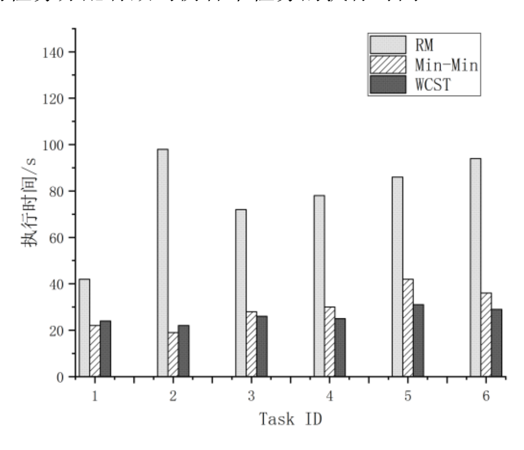


图 2　任务执行时间对比图

Fig.2　Task execution time comparison chart

能量消耗是影响用户体验的另一重要因素，图 3 是 RM 算法、Min-Min 算法以及 WCST 算法分别执行六个任务的能耗对比图。从图中可以看出，相比于 RM 算法，WCST 算法能有效降低任务执行的能量消耗，这是因为当智能设备的负载过高或是 CPU 性能不佳时，执行计算密集的任务不仅效率低而且会消耗较多能量，WCST 算法充分考虑了这些因素，从而能够实现更高效的任务卸载，并且相比于 Min-Min 算法，WCST 算法也能在一定程度上降低能耗并实现负载均衡。

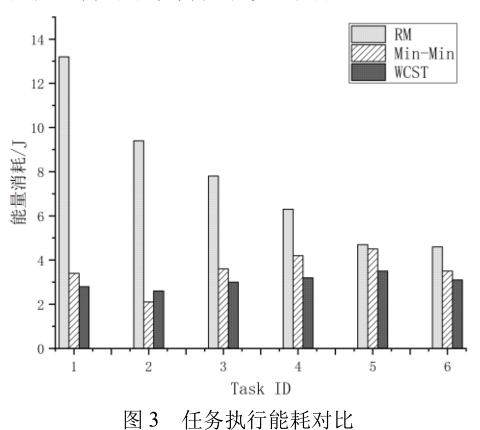


图 3　任务执行能耗对比

Fig.3　Task execution energy consumption comparison chart

鉴于算法中存在大量参数，如 CPU 速度、核心数、负载及节点预测逃离时间等，下面着重针对负载及预测逃离时间增加实验以验证算法中主要参数对任务卸载效率的影响。首先在所提 WCST 算法中去除负载这一因素进行仿真实验，限于篇幅，这里只以任务执行时间作为衡量指标，得到结果如图 4 所示。

之后再针对移动性对任务卸载的影响进行仿真实验，同样在原有算法基础上去除预测逃离时间这一因素，得到结果如图 5 所示。

如图 4 和 5 所示，在原有算法基础上去除负载因素后，相比于原有算法，任务执行时间显著增加，在去除预测逃离时间因素之后，任务执行时间也有些许增加。这说明算法中的各个因素都直接影响着任务执行的效率，并且权重越大的因素对任务执行效率的影响越大。

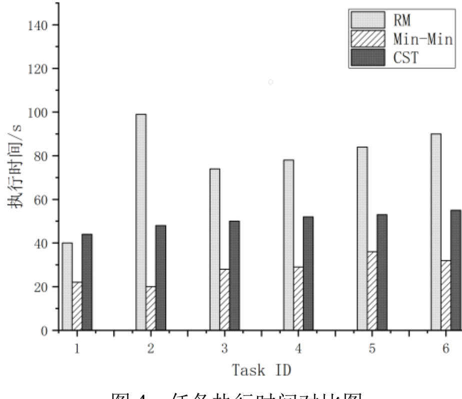


图 4　任务执行时间对比图

Fig.4　Task execution time comparison chart

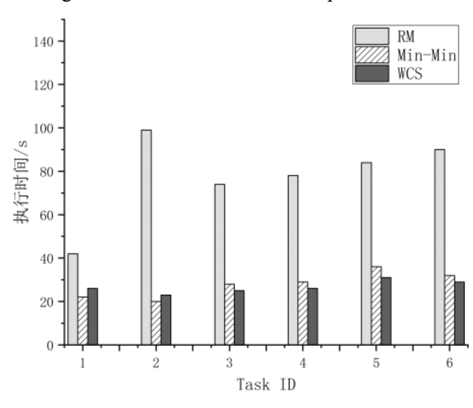


图 5　任务执行时间对比图

Fig.5　Task execution time comparison chart

4　结束语

Ad hoc 云作为移动云计算在无预设基础设施下的一种扩展，具有广阔的应用前景，但 Ad hoc 网络中节点的随机移动性和异构性是影响其应用的重要因素。本文针对 Ad hoc 云中节点移动性和异构性对任务卸载的影响，提出一种基于移动预测的多准则任务卸载算法，根据时间序列分析预测节点逃离时间并将其作为移动性衡量指标，充分考虑节点的异构特性，运用层次分析法得到各因素权重，最后根据得到的组合权重进行任务卸载。仿真结果表明，相比于随机任务分配算法和 Min-Min 算法，所提算法虽然复杂度稍高，但能有效降低任务执行时间和能量消耗，提升用户体验。

参考文献：

[1] Muraleedharan R. Cloud-vision: real-time face recognition using a mobile-cloudlet-cloud acceleration architecture [C]// Computers and Communications. Piscataway, NJ: IEEE Press, 2012: 000059-000066.

[2] Dinhi H T, Lee C, Niyato D, *et al.* A survey of mobile cloud computing: architecture, applications, and approaches [J]. Wireless Communications & Mobile Computing, 2013, 13 (18): 1587-1611.

[3] Liu Yanchen, Lee M, ZhengYanyan. Adaptive multi-resource allocation for cloudlet-Based mobile cloud computing system [J]. IEEE Trans on Mobile Computing, 2016, 15 (10): 2398-2410.



[4] Yaqoob I, Ahmed E, Gani A, *et al.* Mobile Ad hoc cloud: A survey [J]. Wireless Communications & Mobile Computing, 2016, 16 (16): 2572-2589.

[5] Zhou Bowen, Dastjerdi A V, Calheiros R N, *et al.* A context sensitive offloading scheme for mobile cloud computing service [C]// Proc of IEEE, International Conference on Cloud Computing. New York, USA: IEEE Press, 2015: 869-876.

[6] Li Bo, Pei Yijian, Wu Hao, *et al.* Heuristics to allocate high-performance cloudlets for computation offloading in mobile Ad hoc clouds [J]. Journal of Supercomputing, 2015, 71 (8): 3009-3036.

[7] Zhou Bowen, Dastjerdi A V, Calheiros R, *et al.* mCloud: A context-aware offloading framework for heterogeneous mobile cloud [J]. IEEE Trans on Services Computing, 2017, PP (99): 1-1.

[8] Shi Ting, Yang Mei, Li Xiang, *et al.* An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds [J]. Pervasive & Mobile Computing, 2016, 27 (C): 90-105.

[9] Guo Xijuan, Liu Liqing, Chang Zheng, *et al.* Data offloading and task allocation for cloudlet-assisted Ad hoc mobile clouds [J]. Wireless Networks, 2018: 1-10.

[10] Rojas I, Pomares H. Time series analysis and forecasting [J]. Contributions to Statistics, 2016.

[11] Sun Lu. A min-max optimization approach for weight determination in analytic hierarchy process [J]. 东南大学学报（英文版）, 2012, 28 (2) .

[12] Azar H, Majma M R. Using a multi criteria decision making model for managing computational resources at mobile ad-hoc cloud computing environment [C]// Proc of International Conference on Engineering and Technology. Piscataway, NJ: IEEE Press, 2017: 1-5.

[13] Patel G, Mehta R, Bhoi U, *et al.* Enhanced load balanced Min-min algorithm for static meta task scheduling in cloud computing [J]. Procedia Computer Science, 2015: 545-553.